APPLICATION FOR UNITED STATES LETTERS PATENT

For

METHOD AND APPARATUS FOR ESTABLISHING A QUALITY OF SERVICE MODEL

Inventor:

Wolf-Dietrich Weber

Prepared by:

BLAKELY SOKOLOFF TAYLOR & ZAFMAN LLP
32400 Wilshire Boulevard
Los Angeles, CA 90025-1026
(408) 720-8598

Attorney's Docket No.: 2998.P035

"Express Mail" mailing label number: EV 336583672US
Date of Deposit: October 31, 2003
I hereby certify that I am causing this paper or fee to be deposited with the United States Postal Service "Express Mail Post Office to Addressee" service on the date indicated above and that this paper or fee has been addressed to the Commissioner for Patents, P.O. Box 1450, Alexandria, Virginia 22313-1450
Carla Vignola
(Typed or printed name of person mailing paper or fee)

(Signature of person mailing paper or fee)
10-31-03
(Date signed)

METHOD AND APPARATUS FOR ESTABLISHING A QUALITY OF SERVICE MODEL

FIELD OF THE INVENTION

[0001] The present invention generally relates to integrated systems and an aspect specifically relates to guaranteeing quality of service in integrated systems.

BACKGROUND

[0002] A System on a Chip (SoC) is an Integrated Circuit (IC) incorporating most or all of the necessary electronic circuits and parts for a system such as a cellular telephone, digital camera, Set Top Box (STB), etc. The SoC may incorporate several circuits that might otherwise be on individual chips, such as a central processing unit (CPU), direct memory access (DMA) unit, memory, input/output (I/O) circuitry, and other circuits required by the specific application. By including all of the circuitry required for a specific application on one IC, production costs for the system and the size of the system can be reduced, and the reliability of the system can be improved.

[0003] A SoC is a single chip including several interacting entities. The entities may be referred to as Intellectual Property (IP) cores, since they are generally licensed from other vendors, rather than produced by the manufacturer of the SoC. Initiators, such as a CPU, issue requests to targets, such as a memory, for service. For example, a CPU may need access to a portion of a memory. The CPU would issue a request to the memory for the

specific data. The memory would then service the request and return the requested data to the CPU. The initiators and targets are connected through interconnects.

[0004] Quality of Service (QoS) may refer to an expectation of performance in terms of how quickly requests are served. For example, an initiator can issue a request, and can expect those requests to be satisfied by the target within a specific time. Performance may be specified in several different ways. Bandwidth performance refers to receiving a certain number of requests per unit time. Latency performance refers to the time for a certain request to be returned. Jitter performance refers to a variation in the time between requests or responses arriving.

[0005] QoS standards are especially important when using a SoC because initiators used in SoCs typically have very tight service requirements. For example, some initiators (such as CPUs), have tight latency requirements, and need to be served quickly. Other initiators (such as communication interfaces) are more sensitive to bandwidth and jitter performance. Some SoC's suffer from a QoS model that cannot ensure that certain performance guarantees are met in the SoC.

BRIEF DESCRIPTION OF THE DRAWINGS

[0006] **Figure 1** illustrates an example System on a Chip according to one embodiment of the invention.

[0007] **Figure 2** illustrates a system for implementing a QoS model according to an embodiment of the invention.

[0008] **Figure 3** is a flow chart describing a QoS model according to one embodiment of the invention.

[0009] **Figures 4A-C** illustrate an arrival model according to one embodiment of the invention.

[0010] **Figures 5A and 5B** illustrate a service model according to one embodiment of the invention.

[0011] **Figure 6** illustrates an allocation count according to an embodiment of the present invention.

[0012] **Figure 7** is a flow chart illustrating using an allocation count to establish priority for specific threads.

[0013] **Figure 8** is a flow chart describing the operation of an allocation count using an adjustable positive limit.

SUMMARY

[0014] A method and an apparatus for a Quality of Service (QoS) model are disclosed. According to the QoS model, a request is received from an initiator in a first time less than or equal to one less than an ordinal number times an arrival interval, where the ordinal number signifies a position of the request among a group of requests. Also according to the model, the request that has been serviced is returned to the initiator in a second time less than or equal to a constant term plus the ordinal number times a service interval.

DETAILED DESCRIPTION

[0015] In the following description, numerous specific details are set forth, such as examples of named components, connections, number of requests in a group, etc., in order to provide a thorough understanding of the present invention. It will be apparent, however, to one skilled in the art that the present invention may be practiced without these specific details. In other instances, well-known components or methods have not been described in detail but rather as a block diagram to avoid unnecessarily obscuring the present invention. Thus, the specific details set forth are merely exemplary. The specific details may be varied from and still be contemplated to be within the spirit and scope of the present invention.

[0016] In general, methods and apparatus for implementing a Quality of Service (QoS) model are disclosed. According to an embodiment of the invention, the QoS model may be implemented on a System on a Chip (SoC) or other system where it is necessary to guarantee service between an initiator and a target. Initiators and targets are connected through an interconnect. An initiator, such as a Central Processing Unit (CPU) can request service from a target, such as a Random Access Memory (RAM). A thread is a channel to send requests from an initiator to a target. The interconnect, guided by a QoS unit, determines which threads will be serviced by the target. According to the QoS model, a contract may be established between an initiator and the rest of the system. According to the contract, the initiator guarantees that certain requests will arrive by certain deadlines, and the rest of the system guarantees that those requests will be serviced by other deadlines. The QoS model

comprises an arrival model and a service model. The arrival model describes deadlines for receiving requests from the initiator, and the service model describes deadlines for receiving service from the target and the interconnect. According to the arrival model, requests from the initiator must arrive before a time n*a, where 'n' is the number of the request after the first request, and 'a' is a predetermined arrival interval. According to the service model, responses are issued in a time less than K + n*s, where 'K' is a predetermined constant term, and 's' is a predetermined service interval.

[0017] According to another embodiment, an allocation count is maintained for certain threads. The allocation count may track whether a specific thread is being serviced. The allocation count may increment at a regular interval, and decrement when the thread is serviced. The allocation count can be used to determine which threads will be serviced. This determination could be based upon whether the allocation count is currently positive. The allocation count may have a positive limit to ensure that an idle thread is not given too much priority. The positive limit may also be adjustable to insure that a lower priority bandwidth-allocated thread that is not serviced because a higher priority claim is being serviced will eventually receive its allocation.

[0018] **Figure 1** illustrates an example System on a Chip according to one embodiment of the invention. The SoC 10 includes several initiators 11 coupled to several targets 12 through an interconnect 13. Several initiators 11a, 11b, and 11c are shown. A reference simply to an initiator 11 indicates that any of the initiators 11a-c may apply in that instance. The same holds

true for the targets 12a-c, and etc. The initiators 11 may include devices such as CPUs, Direct Memory Access units (DMAs), graphics systems, audio systems, etc. The targets may include devices such as memories including cache memories, Random Access Memories (RAMs), Read Only Memories (ROMs), peripherals, DMA units, register programming interfaces, etc. The initiators 11 generally require service by the targets 12. The interconnect 13 links the various initiators 11 to the various targets 12. Generally, any initiator 11 can request service from any target 12.

[0019] An initiator 11 communicates with a target 12 by issuing requests 14 and receiving responses 15 to and from the interconnect 13. A request 14 may be, for example, a memory read request. The corresponding response 15 would then be data satisfying that request. By sending the response to the requesting initiator 11, the requested target 12 is said to have "serviced" the request. A QoS model describes a contract between an initiator 11 and the rest of the system at the boundary 16 between the specific initiator 11 and the interconnect 13. The QoS model specifies deadlines for receiving requests from the initiator 11 and deadlines for returning responses to the initiator 11.

[0020] **Figure 2** illustrates a system for implementing a QoS model according to an embodiment of the invention. The system 20 may be a SoC or other system requiring QoS management. Several initiators 11a-c are communicating with a target 12a. Although the target 12a is specified here, it is understood that any target 12 may be used. The initiators 11a-c issue groups of requests that are to be serviced by the target 12a. Each initiator 11a-c issues requests on one or more threads 21a-c. A thread 21 is virtual channel

established over a physical channel 22. As shown here, each initiator 11a-c has its own dedicated physical channel 22a-c that corresponds to that initiator 11a-c. Several threads may be multiplexed on the same physical channel 22. Requests from different threads are received at one or more arbitration points 23 inside the interconnect 13. The arbitration point 23 may be specific to the target 12a. In another embodiment, the arbitration point 23 may serve several different targets 12. The arbitration point 23 determines when, and in which order, requests are presented to the target 12a, and the target 12a determines its service timing, and in a multi-threaded case, some of the service ordering. When the requests have been serviced, they are returned to the initiators 11 as threads of responses 24a-c along the return channels 25a-c. The responses have been serviced by the target 12a, and are directed back to the original initiator 11 by the split point 26.

[0021] Several threads are shown on the physical channels 22 and 25. The threads 21a-c are request threads from the initiators 11a-c which are virtual channels carrying requests from the initiators 11a-c. Although they are only shown on a portion of the physical channels 22, the threads 21 "virtually" extend the length of the physical channels 22. Likewise, the response threads 24 are virtual channels along the physical response channels 25. A request channel 26 sends requests from the interconnect 13 to the target 12a. The request channel 26 is also observed by a QoS unit 27. The QoS unit 27 is coupled to the request channel through a channel 28a, and to the arbitration point 23 through the channel 28b. A response channel 29 sends responses from the QoS unit to the interconnect 13. As can be seen, the threads 21a, 21b,

and 21c are multiplexed on the request channel 26. Likewise, the threads 24a, 24b, and 24c are multiplexed on the response channel 29.

[0022] The QoS unit 27 issues instructions to the interconnect 13 through the channel 28b. The QoS unit 27 could be inside the target 12a, inside the interconnect 13, or, as shown, independent. The interconnect 13 ultimately decides which request is issued to the target 12a, but the QoS unit 27 guides the interconnect 13 according to the contract and the QoS model. For each thread 21, the QoS unit 27 determines when the threads 21 will be presented to the target 12a for service according to the various contracts with the different initiators 11. The QoS unit 27, target 12a, and interconnect 13 together satisfy the requirements of the QoS contracts. The specifics of these contracts will be discussed below.

[0023] **Figure 3** is a flow chart describing a QoS model according to one embodiment of the invention. In one embodiment the QoS unit 27, the interconnect 13, and the targets 12 implement the process 30. Block 31 describes an arrival model. In block 31, a group of requests is received from an initiator. Each request, according to the QoS model, is received at a time less than or equal to n*a, where n is an ordinal number describing the number of the request after the first request. For example, *n* is 0 for the first request, 1 for the second request, etc. The variable *a* refers to an arrival interval, which is a predetermined time that refers to the system's accepted interval for request arrivals. According to this model, a group of seven requests arrives before a time 6*a* after the first request arrives. Each request in the group can arrive at a time *n*a* after the first request, however the request may also arrive

at any time before. For example, the second request can arrive any time after the first request and before *a*, the third request any time after the second request and before 2*a*, etc. The model establishes a deadline at or before which each request must arrive. The arrival interval *a* may be fixed by the system or variable, depending on the application. It is understood that a first logic can determine whether the arrival model is satisfied.

[0024] Block 32 describes a service model. In block 32, a response is sent to the initiator in a time less than or equal to $K + n*s$. The K term is a constant term that covers such quantities as the latency and jitter of service. The K term is added to each group, and gives a target servicing the group more latitude to schedule other groups that may have higher priority or to lead to higher overall system efficiency. The *s* variable is a service interval that is analogous to the *a* variable. The K and *s* terms can be fixed for a specific system, or may change depending on the initiator 11, the target 12, etc. It is understood that a logic, including the target 12, the interconnect 13, and the QoS unit 27 can satisfy the service model once it is determined that the arrival model has been satisfied.

[0025] The QoS model comprises two parts: an arrival model and a service model. **Figures 4A-C** illustrate an arrival model according to one embodiment of the invention. **Figure 4A** illustrates a group of requests according to an arrival model 40. The arrival model 40 includes a time line 41 to indicate when the requests 42 are received. The group of requests 44 includes several individual requests 42. The model 40 includes a series of deadlines by which the requests 42 should be received in order to satisfy the

contract. The request interval 43 or *a* signifies a deadline by which time the request should be received by the rest of the system. For each request in the group, the request must arrive before a time established by the following equation:

$$n*a$$

where n corresponds to the number in sequence after the first request in the group (e.g., the third request 42c has an n=2). The arrival time of the first response may be defined as time 0.

[0026] A group of requests 44 comprising the requests 42a-g may be sent by the initiator 11 to the interconnect 13. The initiator 11 issues a group of requests 44 to a single target 12. According to the QoS contract, the entire group of requests 44 is received by the rest of the system in a time less than or equal to one less than the number of requests in the group times the request interval *a* 43. Each individual request is received before *a* times one less than the number of the specific request. For example, the second request 42b is received before a time 1*a*, and the third request 42c is received before a time 2*a*. As can be seen, each individual request in the group 44 arrives in a time less than or equal to $n*a$, and the QoS arrival model is satisfied for the group of requests 44. It can be further seen that each individual request in the group 44 arrives just before its arrival deadline $n*a$. Such behavior is characteristic of isochronous data production processes, which are common in applications such as telecommunications and streaming media.

[0027] **Figure 4B** illustrates two request groups. The request group 44 has been divided into two request groups 51 and 52, which may be necessary

because requests 42d-g do not satisfy the arrival model when grouped with 42a-c. The request group 51 comprises the requests 42a-c, and the request group 52 comprises the requests 42d-g. In order to satisfy the arrival model, the request 42c must arrive before $2a$ after the request 42a arrives. Likewise, in order to satisfy the arrival model, the request 42g must arrive before $2a$ after the request 42d arrives.

[0028] **Figure 4C** illustrates a received request group. The request group 61 comprises the requests 42a-g. According to the QoS model, the request 42b must arrive before time a, the request 42c must arrive before time $2a$, etc. As can be seen in **Figure 4C**, the request 42f is received before the time $2a$, even though the deadline for receiving the request 42f is time $5a$. The model 60 illustrates that an initiator 11 is free to send requests early. The request 42g is received before the time $6a$. As can be seen there is a large gap between the time that the sixth request 42f and the seventh request 42g are received. However, since the requests 42a-g are all received according to the QoS model, the initiator 11 has satisfied the contract.

[0029] **Figures 5A and 5B** illustrate a service model according to one embodiment of the invention. According to **Figure 5A**, the service model 70 includes the group of requests 71 including the requests 42a-d. According to the arrival model, the group 71 is to be received before the time $3a$. The timeline 72 shows the time at which specific requests are serviced. According to the service model, a group of requests must be serviced before a time equal to:

$$K + n*s$$

where *K* is a constant term and *s* is the service interval. The *K* term is a term included in the contract, and applies to each group of requests. The *K* term gives the target 12 extra time to service the group 71. The target 12 may divide up the *K* term as it wishes when servicing the request group 71. The *K* term could be an initial latency term, or can be used by the QoS unit 27 however it desires. The *K* term can be used, for example, to give the target 12 more time to service a request from another initiator 11. In one embodiment, the service interval *s* is a time that is greater than or equal to the arrival interval *a*. Since the requests cannot be serviced until they have arrived, the service interval *s* is necessarily greater than or equal to the arrival interval *a*. If *a* is less than *s*, the service model behaves as though *a* were equal to *s*. In one embodiment, it is desirable to have *a* and *s* equal. The service interval *s* may also be thought of as a nominal bandwidth term. In this example, the *K* term is equal to the service interval *s*, for simplicity. However, it is understood that any *K* term can be chosen independent of the service interval *s*. Also, the service term is given an arbitrary value of $s=1.5a$ in this example.

[0030] According to the service model, the entire group 71, which comprises four requests, must be serviced by the time equal to $K+n*s$, which is $s+3*s = 4s$ (or $6a$). Remembering that the *K* term is assigned a value of s in this example, the first request 42a must be serviced by the time *s*, since $s+0*s = s$. Likewise, the second request 42b must be serviced by the time $2s$, since $s+1*s = 2s$. The interconnect 13 and target 12 has used the *K* term here to delay the servicing of the first request 42a. As can be seen in **Figure 5A**, the group 71

has been serviced according to the model, since each request is received before its respective deadline.

[0031] **Figure 5B** illustrates an alternative servicing of the group 71. In this example 80, remembering that *K* has a value of *s*, the first and second requests 42a and 42b are serviced before the time *s*. The third request 42c is serviced at 2*s*, and the fourth request 42d is not serviced until 4*s*. The fourth and last request does not need to be serviced until 4*s* according to the model, and since the target 12 has serviced the first three requests 42a-c early, the target 12 is free to service other requests as long as the final request 42d is serviced before a time 4*s*. As can be seen in **Figure 5B**, the group 71 has been serviced according to the model.

[0032] **Figure 6** illustrates an allocation count according to an embodiment of the present invention. An allocation count 90 can be maintained in the QoS unit 27 for each thread. The allocation count 90 generally measures whether or not a specific thread is being serviced. In one embodiment, there are three types of threads: 1) high-priority threads which are given priority for service over all other threads as long as they stay within a pre-allocated portion of the target's 12 bandwidth, 2) bandwidth allocation threads which are generally guaranteed a portion of a target's 12 bandwidth, and 3) best effort threads, which are serviced whenever the target 12 has extra bandwidth to do so. Allocated bandwidth and priority threads are monitored using the allocation count 90. The allocation meter 91 is an illustration of the number of credits issued to a specific thread. The allocation meter 91 has a positive limit 92 and a negative limit 93. These limits are explained below.

[0033] The allocation count 90 can be used to determine a priority between threads that are subject to bandwidth allocation (i.e. high-priority and bandwidth allocation threads). Generally, if a thread is not being serviced, often because a high-priority thread has required the service of a specific target 12, the allocation count 90 will become increasingly positive. Conversely, if a thread has received more service than it was allocated, its allocation count will become negative. A negative allocation count 90 can be used to demote the priority of that thread, giving other threads a better chance of receiving service.

[0034] At a regular interval, the allocation count 90 is incremented. For example, at time 0 the allocation count 90 for a specific thread is 0. At time *t*, the thread is issued one credit. Therefore, if the thread has not requested service, the allocation count 90 goes positive, to a count of +1. When the thread receives service, a credit is debited. For example, at time *t*, if the thread requests service once, the allocation count 90 for the thread will be 0, since the thread has received one credit at the time *t* (its regular credit), and has had that credit debited by having its request fulfilled. It is possible for the allocation count to go negative. For example, if at time *t*, a thread has already requested service twice, the thread will merely have received one credit, and will have two credits debited, resulting in an allocation count 90 of –1.

[0035] The allocation count 90 has a positive 92 and a negative 93 limit. As shown here, the positive limit 92 is +7 credits, and the negative limit 93 is –7 credits. If a thread is idle for a long time, the thread will accumulate an excess of credits. As a result, even with the thread resuming requests for service, the

allocation count 90 may never return to zero, and the specific thread may always be serviced. For this reason, the positive limit 92 is established. A large positive limit makes it difficult to honor the QoS contract of other initiators 11, while a QoS model using a small positive limit may not be capable of handling request arrival jitter introduced by the interconnect 13. Further, a higher positive limit may be warranted where the QoS scheme and target behavior introduce service jitter. Hence the need for the dynamic adjustment of the positive limit.

[0036] A negative limit 93 is also established. The negative limit 93 protects a thread from having too many requests serviced and exceeding its allocated bandwidth by too much. If this is the case, the thread may not receive service for a long period of time because it is constantly being demoted as a result of its negative allocation count 90. The negative limit 93 thus reduces service jitter.

[0037] **Figure 7** is a flow chart illustrating using an allocation count 90 to establish priority for specific threads. The process 100 explains awarding priority to specific threads. This priority may be used to determine when a target 12 will service a thread 21. In block 101, it is determined whether there are requests from threads having positive allocation. The positive allocation is determined and accumulated using the technique described relating to the allocation count 90. If threads with a positive allocation are found, in block 102, the highest priority thread among those with positive allocation is chosen. According to one embodiment, the highest priority thread may be a thread that is a high-priority thread or a thread having the most positive

allocation count 90. According to other embodiments, other priority systems may be established.

[0038] In block 103, if there are no threads having positive allocation, the highest priority thread is chosen to be serviced. As above, the priority may be determined using different techniques, including awarding service to a thread having a high-priority designation. The allocation priority may be implemented in the QoS unit 27. Once the QoS unit 27 has determined the thread having the highest priority, that thread is serviced. The process may continue for future service.

[0039] **Figure 8** is a flow chart describing the operation of an allocation count using an adjustable positive limit. The positive limit 92 may need to be adjusted if, for example, a high-priority thread is monopolizing a target 12. A bandwidth allocation thread that needs access to the target 12 may be left waiting for a long period of time because of the high-priority thread. Eventually, the allocation count 90 for the bandwidth allocation thread would reach the positive limit 92. However, the bandwidth allocation thread has still not been serviced. It may be desirable in some instances to dynamically raise the limit 92 in such a situation, because if the limit is left constant, in some instances the QoS contract may not be satisfied.

[0040] The process 110 may be exercised each time a request is to be serviced by a target 12. In block 111, it is determined whether a high-priority request from a high priority thread was serviced while its allocation was positive. If a request was serviced from a high-priority thread, in block 112, the positive limit 92 of all lower-priority threads is increased by an amount

proportional to their allocation rate. For example, if a bandwidth allocation thread has an allocation rate of 50% (i.e., the thread is allocated 50% of the specific target's 12 bandwidth), the limit 92 is raised by 50% of the credit consumed by the high-priority thread. For example, the positive limit 92, in one embodiment, may be 6. Where the high-priority thread consumed two credits, the limit would be raised to 7. Returning to block 111, if no high-priority request was serviced, the process 110 advances to block 113.

[0041] In block 113, it is determined whether a high-priority request received more allocation while its allocation count was positive. If the high-priority thread's allocation count increased while it had a positive allocation count, this is an indication that the higher-priority thread is not requesting service. Since the higher-priority thread is not requesting service, the lower-priority thread can receive service, thus decreasing its allocation count 90. Either the lower-priority thread is being serviced or not requesting service, but in either case the positive limit 92 should be returned to normal in block 114 to avoid giving priority to a thread that does not need it.

[0042] This invention has been described with reference to specific embodiments. It will, however, be evident to persons skilled in the art having the benefit of this disclosure that various modifications and changes may be made to these embodiments without departing from the broader spirit and scope of the invention. The specification and drawings are accordingly to be regarded in an illustrative rather than restrictive sense.